Numerical Calculus I: differentiation

Anthony Lee

December 2024

Outline

Outline

Introduction

- Statistical computations often involve computing derivatives and integrals.
- For theoretical work, we often analyze these objects purely mathematically.
- For practical work, we need to actually differentiate and integrate specific functions.
- In many cases, this can be done exactly but there are many situations where we need to approximate a derivative / integral.
- In some situations we really want an algorithmic solution as a fitting procedure may involve

- creating new functions, and
- doing some calculus with them.

Outline

Derivatives

- The derivative of a function $f : \mathbb{R} \to \mathbb{R}$ is the function f' given by
$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h},$$
if the limit exists.
- Symbolic computations (e.g. in Maple) can compute lots of derivatives, but we are interested here in a different approach.
- Natural approach:
$$f_h'(x) = \frac{f(x+h) - f(x)}{h},$$
with h "suitably small".

Gradients and partial derivatives

- The ith partial derivative of $f : \mathbb{R}^d \to \mathbb{R}$ at a is

$$\frac{\partial f}{\partial x_i}(\mathrm{a}) = \lim_{h \to 0} \frac{f(a_1, \ldots, a_{i-1}, a_i + h, a_{i+1}, \ldots, a_d) - f(\mathrm{a})}{h}.$$

- The gradient of a function $f : \mathbb{R}^d \to \mathbb{R}$ is the vector of partial derivatives

$$\left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d}\right).$$

- These are used a lot and d can be very large indeed.
- We "only" need to approximate derivatives of $f : \mathbb{R} \to \mathbb{R}$.

Cancellation error

- One problem: computers don't do precise calculations.
- They use a large number of real numbers to approximate the whole of $\mathbb{R}$.
- This is annoying! Consider the code in the lecture notes:
  - `a <- 1e16; b <- 1e16 + pi; d <- b-a`
- The value of `d` after these operations is 4, which is quite far from π.
- Example of a problem: taking the difference of two floating point numbers with similar size and sign.
  - Exactly the kind of issue we have with approximating $f_h'(x)$.

Approximation error: part I

- Consider Taylor's Theorem:

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(\xi),$$

for $\xi \in (x, x+h)$.

- Assume $|f''| \leq L$ everywhere, at least to make a point.
- Rearranging gives

$$\left| \frac{f(x+h) - f(x)}{h} - f'(x) \right| = \frac{1}{2}h|f''(\xi)| \leq \frac{Lh}{2}.$$

- Quantifies the error from taking finite h.

Approximation error: part II

- We can't compute f pointwise exactly.
- Suppose we actually compute $\tilde{f}$, which satisfies the relative error criterion:
$$|\tilde{f}(x) - f(x)| \leq \epsilon |f(x)|.$$
- Then if $|f(z)| \leq L_f$ everywhere, we obtain
$$\left| \frac{\tilde{f}(x+h) - \tilde{f}(x)}{h} - \frac{f(x+h) - f(x)}{h} \right| \leq 2\frac{\epsilon L_f}{h}.$$
- So we have bounded the difference between what we compute and $f_h'(x)$.

Approximation error: the tradeoff

- Now we can bound the total error (triangle inequality):

$$\left| \frac{\tilde{f}(x+h) - \tilde{f}(x)}{h} - f'(x) \right| \leq \frac{Lh}{2} + \frac{2\epsilon L_f}{h}.$$

- Minimizing w.r.t. h gives

$$h = \sqrt{\frac{4\epsilon L_f}{L}},$$

which manages the tradeoff between the "rounding error" and not taking the limit.

- This explains why $h \sim O(\sqrt{\epsilon})$ is relatively widespread in practice.

Other finite difference formulae

- There are other schemes, e.g

$$f_h'(x) = \frac{f(x+h) - f(x-h)}{2h},$$

  which can be more accurate.

- Similarly, there are formulae for approximating higher order derivatives:

$$f_h''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

- These can essentially be analyzed the same way.

Outline

Differentiation is “easy”

- Computations can be expressed as compositions of functions involving elementary arithmetic operations.
- In principle, the chain rule and some “primitive” derivatives are sufficient to allow computation of any derivative.
  - This is very different to integration.
- So we could automate it!
  - Symbolic computation is one route, but this tends to lead to complicated expressions quite quickly.
  - Here, we will look at automatic differentiation, a quite different approach.

Automatic differentiation

- Goal: compute derivatives *exactly* as a by-product of using the computer code that evaluates the function.
- Idea: use the chain rule.
- First the univariate case, for simplicity.
- If $y = f_3 \circ f_2 \circ f_1(x)$, then write

$$w_0 = x \quad , w_1 = f_1(x), \quad w_2 = f_2(w_1), \quad y = w_3 = f_3(w_2).$$

- We obtain

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\frac{\partial w_1}{\partial x} = \frac{\partial f_3}{\partial w_2}(w_2)\frac{\partial f_2}{\partial w_1}(w_1)\frac{\partial f_1}{\partial x}(x).$$

- Can compute each partial derivative

$$\frac{\partial f_i}{\partial w_{i-1}}(w_{i-1})$$

when computing $w_i = f_i(w_{i-1})$, and then take the product of these (accumulate).

Automatic differentiation: forward mode

- Now with vectors; assume $f : \mathbb{R}^n \to \mathbb{R}^m$.
- One can lay out the vectors so that

$$\mathrm{w}_i = f_i(\mathrm{w}_{i-1}),$$

  with $\mathrm{x} = \mathrm{w}_0$, $f_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$, $\mathrm{w}_i \in \mathbb{R}^{d_i}$ and $\mathrm{y} = \mathrm{w}_L$.
- Wolog assume we want the partial derivative of y w.r.t. x_1.
- We compute everything forwards: as we compute w_{ij} we also compute

$$\frac{\partial w_{ij}}{\partial x_1} = \sum_{k=1}^{d_{i-1}} \frac{\partial w_{ij}}{\partial w_{i-1,k}} \frac{\partial w_{i-1,k}}{\partial x_1}.$$

Forward mode example

- Consider $y = \exp(x_1 x_2) + x_1^2$, evaluated at $\mathrm{x} = (a, b)$.
- Write $\mathrm{w}_0 = (x_1, x_2)$, $\mathrm{w}_1 = (w_{01} \cdot w_{02}, w_{01} \cdot w_{01})$, $\mathrm{w}_2 = (\exp(w_{11}), w_{12})$ and $y = w_3 = w_{21} + w_{22}$.
- Find $\mathrm{w}_0 = (a, b)$

$$\left(\frac{\partial w_{01}}{\partial x_1}, \frac{\partial w_{02}}{\partial x_1}\right) = (1, 0).$$

Then $\mathrm{w}_1 = (ab, a^2)$ and

$$\begin{aligned}\frac{\partial w_{11}}{\partial x_1} &= \frac{\partial w_{11}}{\partial w_{01}} \frac{\partial w_{01}}{\partial x_1} + \frac{\partial w_{11}}{\partial w_{02}} \frac{\partial w_{02}}{\partial x_1} \\ &= w_{02} \cdot 1 + w_{01} \cdot 0 = b.\end{aligned}$$

Similarly,

$$\begin{aligned}\frac{\partial w_{12}}{\partial x_1} &= \frac{\partial w_{12}}{\partial w_{01}} \frac{\partial w_{01}}{\partial x_1} + \frac{\partial w_{12}}{\partial w_{02}} \frac{\partial w_{02}}{\partial x_1} \\ &= 2a \cdot 1 + 0 \cdot 0 = 2a.\end{aligned}$$

Forward mode example

- Consider $y = \exp(x_1 x_2) + x_1^2$, evaluated at $\mathrm{x} = (a, b)$.
- Write $\mathrm{w}_0 = (x_1, x_2)$, $\mathrm{w}_1 = (w_{01} \cdot w_{02}, w_{01} \cdot w_{01})$, $\mathrm{w}_2 = (\exp(w_{11}), w_{12})$ and $y = w_3 = w_{21} + w_{22}$.
- Now, $\mathrm{w}_2 = (\exp(ab), a^2)$

$$\frac{\partial w_{21}}{\partial x_1} = \frac{\partial w_{21}}{\partial w_{11}} \frac{\partial w_{11}}{\partial x_1} + \frac{\partial w_{21}}{\partial w_{12}} \frac{\partial w_{12}}{\partial x_1}$$
$$= \exp(w_{11}) \cdot b + 0 \cdot 2a = \exp(ab) \cdot b,$$

and

$$\frac{\partial w_{22}}{\partial x_1} = \frac{\partial w_{22}}{\partial w_{11}} \frac{\partial w_{11}}{\partial x_1} + \frac{\partial w_{22}}{\partial w_{12}} \frac{\partial w_{12}}{\partial x_1}$$
$$= 0 \cdot b + 1 \cdot 2a = 2a$$

- Finally $w_3 = \exp(ab) + a^2$ and

$$\frac{\partial w_3}{\partial x_1} = \frac{\partial w_3}{\partial w_{21}} \cdot \frac{\partial w_{21}}{\partial x_1} + \frac{\partial w_3}{\partial w_{22}} \cdot \frac{\partial w_{22}}{\partial x_1}$$
$$= 1 \cdot \exp(ab) \cdot b + 1 \cdot 2a.$$

Forward mode recap

- We've looked at the computation of $\partial y/\partial x_1$ at a particular point.
- You can get, e.g., $\partial y/\partial x_2$ by running another pass, or computing relevant quantities alongside.
- Implicitly, we are applying the chain rule for Jacobians

$$J_h(\mathrm{x}) = \begin{bmatrix} \frac{\partial h_1(\mathrm{x})}{\partial x_1} & \cdots & \frac{\partial h_1(\mathrm{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_m(\mathrm{x})}{\partial x_1} & \cdots & \frac{\partial h_m(\mathrm{x})}{\partial x_n} \end{bmatrix},$$

  i.e. using the identity

$$J_{g \circ f}(\mathrm{x}) = J_g(f(\mathrm{x}))J_f(\mathrm{x}).$$

- Of course, part of the issue is how to compute these things automatically:
- source code transformation / operator overloading.

Reverse mode

- An alternative to forward mode is reverse mode.
- We can use associativity of matrix multiplication to justify computing the other way around.
- To be concrete, assume $\mathrm{y} = g(\mathrm{x}) = f_3 \circ f_2 \circ f_1(\mathrm{x})$, with $f_1 : \mathbb{R}^n \to \mathbb{R}^p$, $f_2 : \mathbb{R}^p \to \mathbb{R}^q$, $f_3 : \mathbb{R}^q \to \mathbb{R}^m$.
- Chain rule:

$$J_g(\mathrm{x}) = J_{f_3}(f_2 \circ f_1(\mathrm{x}))J_{f_2}(f_1(\mathrm{x}))J_{f_1}(\mathrm{x}),$$

  which is a product of 3 matrices of size $m \times q$, $q \times p$ and $p \times n$.
- Forward complexity (right to left): $qpn + mqn$.
- Reverse complexity (left to right): $mqp + mpn$.
- Simple case, $m = 1$ and $p = q = n$. Then forward is $\mathcal{O}(n^3)$ but reverse is $\mathcal{O}(n^2)$.

Reverse mode: memory / optimal computation

- In forward mode, we only need one “layer” to compute the next layer.
- In reverse mode, we need to store the values and partial derivatives until we sweep backwards.
- This can be a large memory cost.
- In practice, one can mix forward and reverse mode.
  - Any sequence of appropriate matrix multiplications is fine.
- The optimal sequence is NP-hard (strange examples) or at least of open complexity.

A caveat

- Consider the function, for positive y,

$$B(y;\lambda) = \begin{cases} (y^\lambda - 1)/\lambda & \lambda \neq 0 \\ \log(y) & \lambda = 0 \end{cases} .$$

- If $f(\lambda) = y^\lambda = \exp(\log y \cdot \lambda)$, we have $f^{(k)}(\lambda) = \log(y)^k y^\lambda$.
- Using this, we can find that

$$\lim_{h\to 0} \frac{y^h - 1}{h} = \log(y),$$

and

$$\frac{\partial B}{\partial \lambda}(y;0) = \lim_{h\to 0} \frac{\frac{y^h-1}{h} - \log(y)}{h} = \frac{1}{2}\log(y)^2,$$

but AD will not obtain this expression using only the evaluation of $B(y;0)$.

Outline

Wrapping up

- Finite differences are a classical and fairly robust way to approximate derivatives of sufficiently smooth functions.
- You get approximate values, trading off two sources of error.
- No problem if computation of function involves conditional statements!
- Symbolic computation (e.g. in Maple or Mathematica) can also be useful.
- You get exact expressions, useful mainly for simple enough functions.
- Automatic differentiation can be used to compute exact derivatives at a point.
- You get exact values.